University
of Southern
California

Micro Benchmark Analysis of the KSR1

Rafael H. Saavedra
R. Stockton Gaines
Michael J. Carlton

ISI/RS-93-394

November 1993

INFORMATION SCIENCES INSTITUTE

310/822-15

4676 Admiralty Way/Marina del Rey/California 90292-66

ISI Reprint Series
ISI/RS-94-394
November 1993

Micro Benchmark Analysis of the KSR1

Rafael H. Saavedra
R. Stockton Gaines
Michael J. Carlton

ISI/RS-93-394

November 1993

University of Southern California
Information Science Institute
4676 Admiralty Way, Marina del Rey, CA 90292-6695
310-822-1511

The research was sponsored by the Advanced Research Projects Agency under Rome Laboratories Contract No. F30602-91-C-0146. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, Rome Laboratories, the U.S. Government, or any person or agency connected with them.

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Reprint Series

4. TITLE AND SUBTITLE
Micro Benchmark Analysis of the KSR1

5. FUNDING NUMBERS
C - F30602-91-C-0146

6. AUTHOR(S)
Rafael H. Saavedra, R. Stockton Gaines and Michael J. Carlton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
ISI/RS-93-394

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Reprinted from Proceedings of Supercomputing '93, IEEE Computer Society Press (1993), 202--213, Portland, Oregon. Also available as a technical report from the USC/Department of Computer Science. August 1993. USC-CS-93-538.

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

A new approach, micro benchmarks has recently been developed. Using this technique, we have analyzed the KSR1, and in particular the "Allcache" memory architecture and ring interconnection. We have been able to elucidate many facets of memory performance. The technique has enabled us to identify and characterize parts of the memory design not described by Kendall Square Research. Our results show that a miss in the local cache can incur a penalty ranging from 7.5 microseconds to 500 microseconds (when a dirty "page" in the local cache must be evicted). The programmer must be very careful in placement and accessing of data to obtain maximum performance from the KSR1; the data presented here will help in understanding the performance actually obtained.

14. SUBJECT TERMS
Shared-Memory Multiprocessors, Memory Hierarchy Performance, Micro Benchmarks, Network Interconnects, Performance Evaluation.

15. NUMBER OF PAGES
15

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Micro Benchmark Analysis of the KSR1[§]

Rafael H. Saavedra

Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
saavedra@pollux.usc.edu

R. Stockton Gaines and Michael J. Carlton

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
{gaines,carlton}@isi.edu

ABSTRACT

A new approach, micro benchmarks, has recently been developed. Using this technique, we have analyzed the KSR1, and in particular the "ALLCACHE" memory architecture and ring interconnection. We have been able to elucidate many facets of memory performance. The technique has enabled us to identify and characterize parts of the memory design not described by Kendall Square Research. Our results show that a miss in the local cache can incur a penalty ranging from 7.5 microseconds to 500 microseconds (when a dirty "page" in the local cache must be evicted). The programmer must be very careful in placement and accessing of data to obtain maximum performance from the KSR1; the data presented here will help in understanding the performance actually obtained.

1. Introduction

The KSR1 from Kendall Square Research is a novel new parallel computer. It is the first commercial machine embodying a scalable all cache form of shared memory architecture. In addition, there are a number of other interesting features of the machine.

We report our observations of the KSR1, obtained by means of a suite of small benchmarks that expose the details of the machine characteristics. We refer to these small benchmarks as micro benchmarks. In section 2 we briefly describe the micro benchmark approach, and its application to parallel machines. The micro benchmark suite has been developed and used to analyze the performance of uniprocessor machines [12, 13]. We describe the architecture of the KSR1 as we understand it in section 3. We have run our standard micro benchmark suite for processor performance, and included the results together with comparative results from two other CPUs of interest. The main focus of our work has been to understand and measure the performance of the KSR1's novel "ALLCACHE" memory, which we have extensively analyzed with a new set of micro benchmarks. This work and the results are described in section 5. Section 6 analyses a set of experiments used to measure the effect of contention in the interconnection network.

2. The Micro Benchmark Approach

Recently, one of us (Saavedra) has explored a new approach to benchmark analysis of computers. This approach has been documented in several papers [12, 13, 14]. The approach was developed in reaction to the use of large applications as benchmarks. Though it is hoped that large applications will be more representative of real workloads than synthetic benchmarks or small kernels, it is not clear what features of a particular system they exercise, or what actually accounts for the differences in the performance of these benchmarks on different machines.

The micro benchmark approach returns to the idea of measuring specific features of the machine. But in contrast to measuring only a few parameters, such as floating point multiply, the approach consists of (1) measuring every observable feature of the machine, and (2) making use of the collected set of data in an integrated way. For uniprocessors, one of the most powerful ways of using the micro benchmarks is to predict the performance of a program on a new computer without first porting the program and measuring the results. This is done by analyzing the program to determine how much use is made of each of the machine features measured by the micro benchmarks, and then using the results of the execution of only the micro benchmark suite on the new computer to predict the running time of the program. Using the approach, it has proven possible to estimate accurately the performance of a wide range of programs, including standard benchmark suites such as Spec and Perfect [16, 3]. Further, the analysis of these programs in terms of the features measured by the micro benchmark suite gives insight into the reasons for the observed performance differences for the programs on different computers.

An important factor in machine performance is cache, memory, and network interconnect behavior. A test has been developed that reveals a great deal about the memory hierarchy behavior (including the network), and a way of displaying the data from this test using a set of diagrams that we have named *Physical and Performance Profiles* (or P^3 diagrams) has been developed. We will explain this test, and the main characteristics of the P^3 diagrams, below. The data thus obtained, together with information about the rate of misses in a program, is factored into the other micro benchmark results as part of the prediction methodology.

This paper reports results of the micro benchmark analysis of the KSR1. The machine contains many features (described below) not found in other machines. To understand the performance implications of these features, it has been necessary to develop additional micro benchmarks beyond the initial, general purpose suite. These are described later.

An open and interesting question is the analysis of parallel programs, and the prediction of their performance through the use of micro benchmarks. This involves developing new tests for synchronization, access to shared variables, and probably features provided in run time libraries for parallel machines.

§ This research was supported by the Advanced Research Projects Agency under Rome Laboratories Contract F30602-91-C-0146.

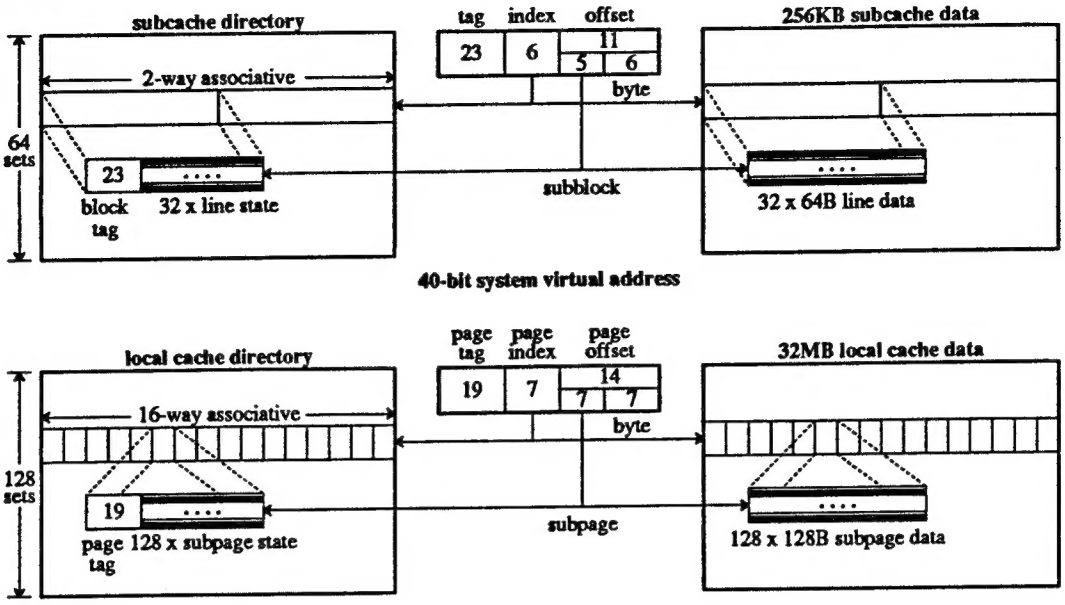

Figure 1: KSR1 cache and subcache organization.

However, it is not clear at this point which factors are relevant to take into account in building a reasonable model of program execution which can produce predictions about the execution time of parallel programs. Furthermore, we believe that a substantial experimental understanding about the performance regimes exhibited by shared memory machines is required before any such model can be developed. We will report separately our results for message passing machines, and for other shared memory machines such as the Stanford DASH [8], for which we have also developed new micro benchmarks.

As will be seen below, our approach reveals a great deal of interesting information about the KSR1. There is much more work to be done, however, to extend the approach into the area of prediction.

3. Architecture of the KSR1.

The KSR1 is a new architecture for parallel machines, and one that attempts to solve the problem of scalability in shared memory multicomputers. The problem is that as the number of processors increases, the average cost of access to memory goes up. One approach has been to design faster memory interconnects, together with highly interleaved memory. This approach appears to be limited to at most a few hundred processors.

An alternate approach, explored by both Kendall Square and the Stanford DASH project, is to use a directory-based caching scheme with a relatively large amount of memory close to each individual processor. Both of these machines use a "message-passing" interconnect mechanism rather than more typical memory interconnect methods. In the case of the KSR1, it is a ring of rings [6], while the Stanford DASH uses a 2 dimensional mesh interconnection [8].

The KSR1 is organized as a ring of rings, with up to thirty two processors connected to each lowest level ring, called ring:0. Ring:0 rings are interconnected by another ring, ring:1. Each node consists of a CPU, a 512-KByte subcache, 32 MBytes of cache memory, a cache directory, a ring interface, and an I/O interface. The directory supports the cache coherency protocol between processors and between rings. A total of 1088 processors can be connected as 34 ring:0 rings interconnected by one ring:1 ring. Kendall Square Research intends to extend the hierarchy so as to connect even more processors in future designs.

The processing component has four functional units: integer, floating point, control, and an I/O unit. Instructions are issued in pairs: an integer or floating point instruction paired with a control or I/O instruction. The machine is a load/store architecture, with loads and stores issued by the control unit. Some floating point instructions result in two floating point operations. There are 32 integer registers, 64 floating point registers, and 32 addressing registers. The integer and floating point registers hold 64 bits, the addressing registers are 40 bits. The machine operates at 20 megahertz and is fully pipelined, with two branch delay slots.

The user view of memory (called the *Context Address* - CA) is a segmented address space. Segments can range from 2^{22} to 2^{40} bytes (in the current implementation) in length. The processor produces 40 bit addresses, interpreted as a segment number and offset. The processor contains an instruction segment table, with 8 entries, and a data segment table of 16 entries.

The addresses generated by the processor are translated into *System Virtual Address* (SVA) space. The segment tables include the base location of the segment in SVA, the segment's length, and access permissions. Presumably, the segment tables are fully associative.

3.1. KSR1 ALLCACHE Memory Architecture

The memory in each processor is organized as a two-level cache hierarchy. There is a large *local cache* that combines the functions of memory and a second level cache, and a *subcache* that is the first level cache. Both caches are managed by a directory structure consisting of a large unit of allocation in the cache directory, and a smaller unit of data transfer, as shown in Figure 1. This is in con-

trast with more common cache organizations in which the units of allocation and transfer are the same.

There are instruction and data subcaches, each 256K bytes. Each subcache is managed by a subcache directory with 128 blocks each containing 32 subblocks. The directory is organized as 64 two-way associative sets with a random replacement policy. A reference to a new subcache block incurs the overhead associated with invalidating all the subblocks of the block being displaced. The subcache is cache coherent with the local cache. The subblocks each contain 64 bytes of data. A read from the subcache is satisfied in 3 clock cycles (load instructions have 2 delay slots).

There is a 32 megabyte local cache in each node. There is no memory with a fixed address (hence the term "ALLCACHE"). Instead, memory is managed through a directory structure that makes all the cache memory in the machine visible and accessible to every processor. A similar memory architecture has been described in [5], where the term Cache Only Memory Architecture (COMA) is used.

The local cache is 16-way set associative, and is organized in 16K byte "pages". Each page is divided into 128 "subpages" of 128 bytes. These subpages are the unit of memory coherence. The directory contains an entry for each of the 2K pages that comprise the memory. Each entry includes a tag and the state of each subpage, of which the visible states are invalid, read-only, exclusive, and atomic. Exclusive means that this is the only copy in any processor's local cache. Atomic is exclusive and locked.

Neither the whole 32 MBytes nor the 16-way associativity is accessible from a user program. The OS sets aside a significant number of pages (close to 30% of the total memory) for its own use and these cannot be displaced out of the cache. We discuss in detail the effective associativity experienced by user programs in section 5.5.

A consequence of this design is that if the processor references a subpage from a new page, all the lines from another page present in the cache may need to be evicted. In order to keep at least one copy of all pages currently being referenced, each page is assigned a "home" node. The home page provides space for every subpage, even if there are no valid subpages at the home node. Having a home node greatly simplifies evicting a subpage, as there is guaranteed to be a node with room for the subpage. If the home node must evict a page, the operating system will swap the page to disk or to another node's local cache to make room for the new page. A significant fraction of the node's memory is set aside by the operating system to be used as home pages [2].

The instruction set includes instructions to prefetch and poststore subpages. A prefetch allows the processor to read a subpage from another node without having to stall while the request is serviced. There can be up to four prefetches outstanding at any time. If this limit is reached, then additional prefetches are either discarded or the processor is forced to block until one of the four pending prefetches completes. A field in the prefetch instruction determines the action to follow. An additional field indicates the state of the subpage that is to be read: exclusive or read-only. A poststore instruction causes the local cache to broadcast a read-only copy of a subpage. All nodes with the subpage's page allocated in their cache will read it, provided the cache directory is not busy.

A multi-ring machine contains a ring interface in each ring:0 ring that is a directory for the entire ring (that is, it contains an entry for every page that is in any processor cache in the ring).

The data rate of ring:0 is 1 gigabyte/second. Ring:1 supports a "fat" structure with multiple rings to provide 1.2, 2.4, or 4.8 Gbytes/s bandwidth. Increasing the number of subrings in a ring:1 structure reduces the total number of available nodes that can be used for processing in ring:0.

The KSR1 provides sequential consistency, which implies that writes to a subpage cannot complete until all other copies present in the machine have been invalidated [7].

3.2. Paging on the KSR1

The KSR1 scheme of allocating space in the caches in large, page-sized units and filling in units of cache lines (subpages) appears to be a reasonable compromise. If we consider the amount of storage required for cache directory information we see that the current implementation will require only about 100KB. This is calculated from having 2048 page entries, each of which includes an 19-bit tag and at least 3 state bits for each of the 128 subpages. A simpler implementation that separately allocated each subpage would require more than 700KB of directory storage.

The cost of this method is that an entire page must sometimes be evicted due to a subpage miss. There are 2^{19} pages in SVA that index to the same set of page positions (see Figure 1), and the cache is 16 way set associative at the page level. So sequential references to a set of only 17 pages can (in the worst pathological case) cause a page eviction at every reference.

An advantage of the paged scheme is that it matches disk accesses well. When accessing a disk, a system wants to get big chunks from the disk since it's slow. With local cache pages, KSR1 can quickly clear a page's state to make room, especially since the disk page is the same size. Without cache pages a system would have to clear each block individually. This is likely to be done serially, since the tags would be in sequential RAM locations.

The KSR1 implementation saves a significant amount of storage per node. This also greatly affects the ring interface node which must duplicate the entire state information of all 32 nodes on a ring:0. This is an important consideration when looking at larger configurations of the KSR1, as the ring directory must not become a bottleneck if the system is to be scalable. More important than the amount of storage in the ring directory is the time and cost to search it: it requires checking each of the 512 (32 caches * 16-way associativity) entries which might have a copy of the referenced page.

3.3. Comments on COMA and NUMA

The COMA organization of the KSR1 contrasts with a more traditional directory-based NUMA (non-uniform memory access) machine such as the Stanford DASH by treating all of its main memory as a cache. NUMA machines treat a memory address as having a static, known location. The distance between a processor and the various main memory modules differs, leading to the non-uniform access distances. Both architectures commonly use the cache block as the coherency unit (i.e. sharing occurs on a block basis). Note that the KSR1 and Stanford DASH both include caches below the level of main memory to improve performance. The DASH has first and second-level caches, while the KSR has its subcache.

One important difference in the architectures occurs when accessing shared data. When a NUMA misses in its cache for consistency reasons, it will initiate a transaction to a node which is functioning as the "home" for this address. However, a COMA machine cannot direct its transaction to a specific location, instead it issues a request that will search the caches of the system until it finds the requested data. Determining which organization will be faster for a particular access depends upon the relative locations of shared data and the details of the coherency protocol.

Factor	KSR1	Alpha	DASH
Integer Add	44.0 ns	6.1 ns	33.9 ns
Integer Multiply	92.5 ns	98.1 ns	360.5 ns
Integer Divide	4968.8 ns	198.9 ns	1023.2 ns
F-Point Add	52.5 ns	22.9 ns	88.2 ns
F-Point Multiply	22.1 ns	20.0 ns	147.2 ns
F-Point Divide	1760.6 ns	171.9 ns	702.4 ns
Complex Arith.	3199.1 ns	182.7 ns	780.3 ns
Intrinsic Func.	7969.6 ns	1134.3 ns	2683.0 ns
Logical Ops.	148.5 ns	27.2 ns	106.7 ns
Branch/Switch	148.2 ns	15.0 ns	36.4 ns
Proc. Calls	757.2 ns	62.0 ns	379.6 ns
Array Indexing	47.3 ns	39.7 ns	111.4 ns
Loop Overhead	101.6 ns	19.5 ns	105.6 ns

Table 1: Single CPU Performance of the *KSR1*, DEC Alpha 4000/610, and *DASH*.

4. Summary of KSR1 CPU Micro Benchmark Measurements

As mentioned above, a micro benchmark suite that measures CPU performance has previously been developed and used to obtain measurements on a variety of uniprocessor machines. Results for many systems have been reported in [12, 13]. We have run the same suite on the KSR1 CPU.

The CPU micro benchmarks are machine-independent, so instead of measuring machine instructions they measure operations defined in a high level abstract machine. The abstract machine is based on the Fortran programming language, so applications written in Fortran compile directly into the abstract machine code. The number and type of operations is directly related to the kind of language constructs present in Fortran. Most of these are associated with arithmetic operations and trigonometric functions. In addition, there are parameters for procedure call, array index calculation, logical operations, branches, and do loops.

In Table 1 we present the results we obtained, together with the results of running the same micro benchmarks on the Stanford DASH and a DEC Alpha 400 model 610 system running at 160 MHz. The processor in the Stanford DASH is the MIPS R3000 running at 33 MHz. In future work, we plan to use these benchmarks and provide a comparison of the KSR1 with a number of other parallel machines.

5. Analysis of the KSR1 Memory Architecture

As discussed above, a general methodology for analyzing the memory behavior of machines with caches has been described in [13]. We have extended this methodology with additional benchmarks that measure the memory hierarchy behavior of shared memory multiprocessors. Here, we give a brief explanation of the approach, and present the results we have obtained for the KSR1.

5.1. Methodology

There are many specific measurements one can make of a memory system. In general, there may be several levels of cache in addition to the main memory in the system. The main memory may be a single global module or distributed among the nodes. If the memory is distributed, the processors may treat each module as local memory, or may share the memory of all modules in a global, shared address space. The properties of the memory interconnect, including bandwidth and latency under a variety of loads, are of interest. There may also be a write buffer associated with each level of cache. There may be separate cache coherency directories as well as the cache itself. It is a challenge to simply measure the performance of all of these mechanisms in a way that shows what happens under a variety of conditions. It is clear that a few simple numbers are far from sufficient to characterize memory behavior.

Beyond the issue of obtaining measurements that characterize the behavior of the memory architecture under a full range of conditions, there is the problem of presenting the results in a more meaningful form than a large table of measurements, or reducing the results to a few average numbers. Most useful would be a presentation of the results that allows a programmer with a specific application to understand what the memory performance of his program would be. We have developed a method of displaying the results that captures a significant amount of information in graphical form. We called these diagrams *Physical and Performance Profiles* (P^3 diagrams) of the memory subsystem as they contain the physical characteristics of each memory structure in addition to the performance characteristics.

5.2. The Structure of the Physical and Performance Profiles

Because of the complexity of the memory architectures of interest, the P^3 diagrams require some effort at interpretation, but they have a great advantage as compared with a set of tables of results, and provide far more information than averages which summarize the measurements. The P^3 diagrams are a set of plots representing the average execution time needed to read, modify and write (a R-M-W cycle) a single element in a sequence of locations (not necessary contiguous) taken from a region of memory as a function of the size of the region (R) and the distance (stride S) between consecutive elements. An alternative experiment consists of reading elements without changing their values. We refer to this type of experiment as read-use cycle (R-U cycle).

The access times are measured by timing the execution of a Fortran loop. Each data point on a curve is the mean time per iteration calculated from performing a fixed number of accesses to an array of the given size, using that stride. The clock resolution of the machine is 20 μsec. By factoring out loop overhead and averaging over a large number of iterations, we believe that the error in our results is generally less than a clock cycle.

Depending on the relative magnitudes of R and S of a R-M-W experiment with respect to the size, width, and associativity of the structures forming the memory hierarchy a distinctive value for the average execution time is obtained. All results are depicted as a set of curves, where each curve corresponds to a particular value of R, with all values of $S=2^n$ from 1 to $R/2$ plotted. In this section we briefly explain how to read these diagrams. A more extensive discussion can be found in [14].

For explanatory purposes, the discussion will focus in the effects of our experiments on a memory hierarchy consisting of a single cache. The explanation extends trivially to more complex hierarchies and in what follows we provide some comments in this respect. Depending on the values of R and S with respect to the size of the cache C, the line (block) size b and associativity a, we can observe one of four basic regimes. Furthermore, the response of a more complex memory hierarchy is just the superposition of the memory structures' individual responses, which always fit one of the four basic regimes.

In the presentation we assume that all variables take values that are powers of two. However, if one of the physical dimensions of a memory structure happens not to be a power of two, it will be necessary to use different sequences of values for R and S.

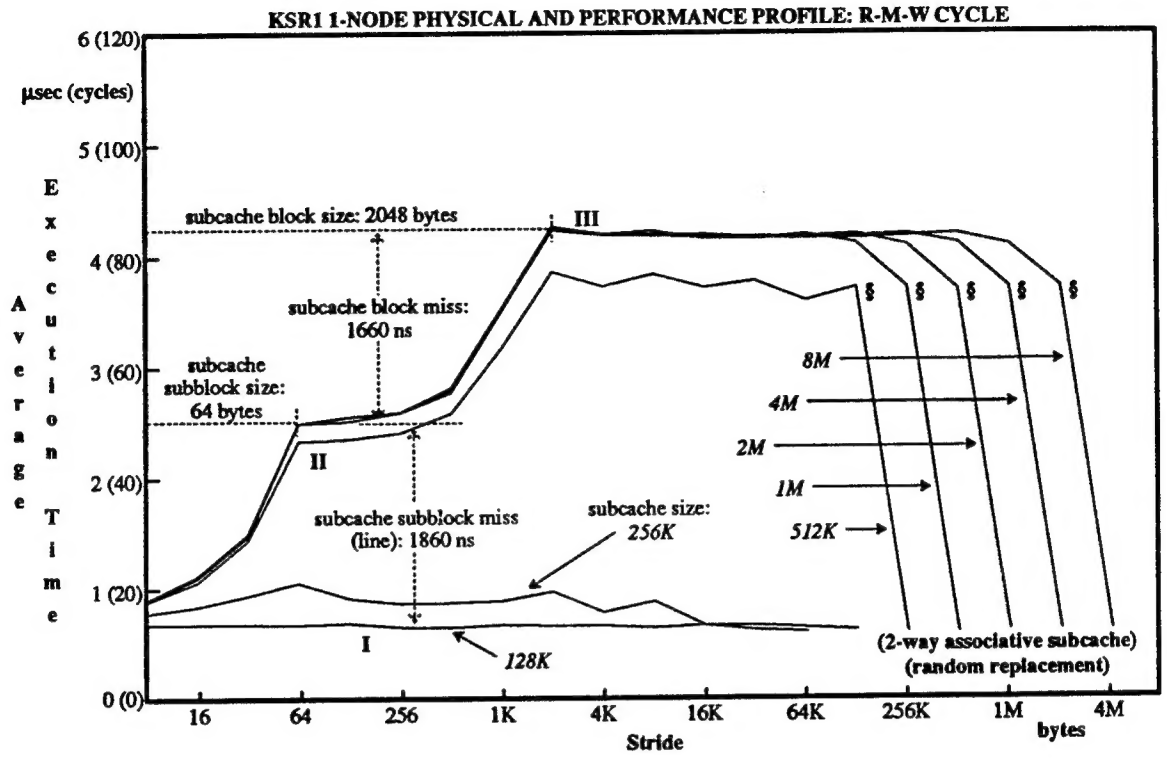

Figure 2: KSR1 single node Read-Modify-Write cycle physical and performance profile.

Each micro benchmark consists of making multiple passes over an array of size R, accessing every S^{th} element. The first pass (at stride 1) over region R will incur some cold misses, but this error is negligible due to the length of the micro benchmark. Micro benchmarks at larger strides will incur no cold misses, as they touch a smaller set of elements.

The simplest regime (regime 1) occurs when $R \leq C$. Here, independently of the stride, all elements accessed by the experiments fit in the cache, so there are no cache misses in the steady-state phase of the experiment. Therefore, the average time of the R-M-W cycle as a function of S is a constant line. Curve $128K$ in Figure 2 is a clear example of regime 1[1].

When $R > C$, misses start to occur, and depending on S we can observe one of three regimes (2.a, b, c). Regime 2.a occurs when $S < b$. Here, there are several consecutive accesses to each cache line in between corresponding misses, so the cache miss penalty is amortized amongst the accesses. As S grows, the average time for the R-M-W cycle increases in proportion to S. All curves in Figure 2 where $R \geq 512K$ and $S \leq 64$ correspond to regime 2.a. Regime 2.b represents the situation where each reference falls into a different cache line and it always generates a miss. Formally, this is true only if the cache replacement policy is either FIFO or LRU. For a random replacement policy, the effect rapidly converges to that of LRU and FIFO as the number of lines mapping to a set increases above the degree of associativity. Regime 2.b occurs when $b \leq S < R/a$. Here, each experiment touches a subset of all cache sets, but the number of cache lines mapping to a set is greater than the associativity. This result follows from the following argument. There are C/ab sets in a cache. In general, an experiment touches $R/(b\lceil S/b \rceil)$ cache lines which are mapped into $C/(ab\lceil S/b \rceil)$ sets if $S \leq C/a$ or into a single set if $S > C/a$. In regime 2.b, $S \geq b$, so S/b is always a whole number greater than one. Therefore, the number of lines touched are R/S and these are mapped into either C/aS sets or a single one. In both cases, each set receives Ra/C or R/S lines, respectively, and it follows from condition $R > C$ that $Ra/C > a$ and $R/S > a$.

Therefore, in regime 2.b, the average time for the R-M-W cycle as a function of S is constant, assuming there are no other effects produced by the other memory structures. In Figure 2, regime 2.b corresponds to the two plateaus present in all curves in the regions $R \geq 512K$ and $64 \leq S \leq 256$, and $R \geq 512$ and $2K < S < R/4$. The last regime (2.c) occurs when the number of different cache lines mapping into the same set is less than or equal the set-associativity. This situation is characterized by condition $R/a \leq S < R$. For this regime, the R-M-W cycle average time drops to the level of regime 1. Furthermore, the ratio R/S at which the drop occurs gives the set-associativity of the memory structure. In Figure 2 all curves where $R \geq 512K$ exhibit this behavior at their rightmost point, indicating that the set-associativity is two.

In the next section, we discuss some specific performance characteristics of the KSR1 that are observable from the P^3 diagrams our

[1] Figure 2 represents the superposition of the effects of two memory structures: the subcache subblock and block organizations. All four regimes, however, are clearly identifiable in the figure and we make reference to it to illustrate the regimes.

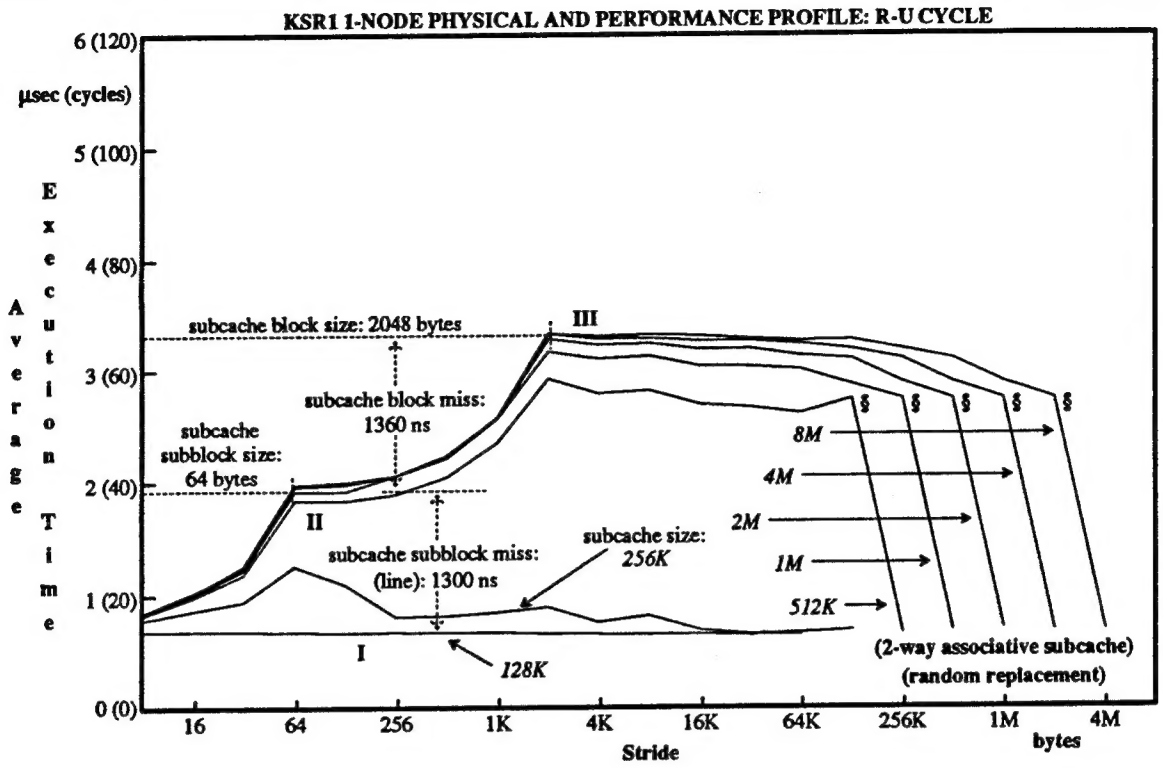

Figure 3: KSR1 single node Read-Use cycle physical and performance profile.

experiments generate. In the diagrams we identify the KSR1 regimes by using roman numerals instead of the basic regime numbers. We do this to avoid confusion, because most of the KSR1 regimes represent the superposition of several basic regimes affecting different memory structures in the memory hierarchy.

5.3. KSR1 Single Ring, Single Node Performance Results

In Figure 3 we show the performance of a single KSR1 node while reading data from the cache. The figure consists of curves for regions of size R=128K to 8MB, with strides of 8 bytes, 16 bytes, ..., R/2 bytes. The KSR1 word size is 64 bits, so 8 bytes is the smallest stride.

When the data set being accessed is smaller than the size of subcache (regime 1) there will be no cache misses and we can read the base time per iteration. The flat curve for the 128KB data set in Figure 3 shows this case, and we see that the average time per iteration of the loop for all strides used was about 650 nanoseconds. This is the time to perform one iteration of the loop with a floating point add and multiply.

The size of the largest such curve with no misses tells us the size of the subcache. In this case we see that the subcache is 256KB. The line is not completely flat due to interference from other data used by the process – the data set is the same size as the cache and any accesses to other data will cause cache misses.

The 512KB and larger curves show us what happens in regimes 2.a, 2.b, and 2.c. The data is initially not in the subcache and the first reference to a subblock will cause a cache miss; succeeding references to the same subblock will hit. At stride 8, there will be one miss and 7 hits. At stride 16, there will be one miss and only 3 hits to each subblock. As we increase the stride, we decrease the number of hits and the cost of the miss is amortized over fewer accesses. At a stride of 64 bytes, the curve flattens out, as every reference is made to a different subblock. This indicates the transition from regime 2.a to 2.b and we are able to conclude that the subblock size for the subcache is 64 bytes.

We can also read the time taken to satisfy a miss to a subblock by measuring the difference in times between the case with a miss on every access (a data set of 512KB and stride of 64) and the case with no misses (a data set of 128KB). We see on the KSR1 that this is approximately 1300 nanoseconds.

Between stride 64 and 2048 the curves repeat the same pattern of rising access times. This regime shows the effect of accessing a new block in the subcache. There is a second major inflection in the curve at a stride of 2048 bytes; this corresponds to the case in which every reference is to a new block of the subcache. From these curves, we are able to deduce that there is a directory structure with blocks and subblocks which manages the subcache (which we call the subcache directory - Kendall Square Research has not published any information about this aspect of the architecture).

At a stride of 256K bytes, the 512K byte curve shows that the cost of a read is the same as the cost when there are no subcache misses. In contrast, if the stride is 128K bytes, the cost of a read includes the cost of a subcache page miss. From this, we can conclude that the subcache is two-way set associative because at a stride of 256K bytes only 2 different subblocks are being accessed and they map to a single set in the cache.

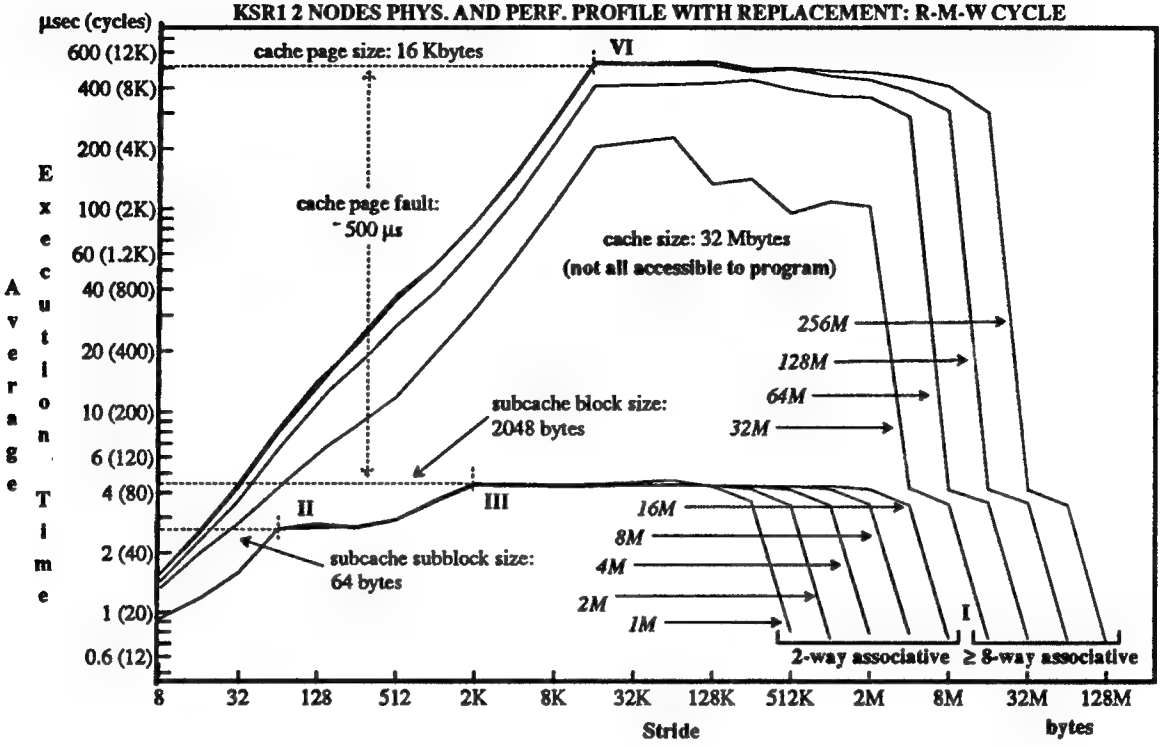

Figure 4: Physical and performance profiles of the KSR1 subcache and cache structures obtained using 2 competing nodes running R-M-W based experiments. Each regime identifies the mean execution time for a particular combination of subcache and cache miss penalties. For array sizes smaller then 16MB, there is no contention between the nodes.

Figure 2 shows the case of a loop that reads a word, modifies it, and writes the result back to memory. Compared with the read case shown in Figure 3, there are extra costs because the blocks of the subcache have been modified. This increases the subcache subblock miss penalty by about 560 nanoseconds (approximately 11 clock cycles), and the cost of a subcache block miss increases by about 300 nanoseconds (6 clock cycles). We note that the base case is 650 nanoseconds, as it was for the read case. From this we conclude that the write is completely overlapped with the loop branch and other operations.

5.4. Subcache Random Replacement Policy

In Figure 3 we can also observe the fact that the subcache replacement policy is random. This manifests itself in the height of curve 512K which reaches a lower height than the other curves in regime III. Regime III corresponds to basic regime 2.b. This basic regime assumes either an LRU or FIFO replacement discipline to enforce that every reference generates a miss (a subcache subblock and block misses in this case). With random replacement, however, some subset of the references will not cause misses.

As mentioned in section 5.2, in a cache of size C and associativity a, an experiment covering a region R will map $Ra/C > a$ cache block into the same set. Now, in a random discipline, the probability p_{surv} that a cache block will remain in the cache after a pass through all elements in the experiment is

$$p_{surv} = \left[1 - \frac{1}{a}\right]^{\frac{aR}{C}-1}. \qquad (1)$$

Consequently, the average execution time per R-M-W cycle in regime III (T_{III}) should be:

$$T_{III} = T_{no-miss} + (1 - p_{surv}) \cdot D_{miss}. \qquad (2)$$

where $T_{no-miss}$ and D_{miss} are, respectively, the average execution time without misses and the miss delay penalty. Now, if we replace the KSR1 parameters in eqs. (1) and (2) we get that the effective subcache miss delay penalty for curve 512K should be $7/8 = 0.875$ of D_{miss}. The results in Figure 3 for curve 512K exhibit an effective delay penalty in the range .86 to .88.

A more subtle manifestation of the random replacement policy in Figures 2 and 3 is the decrease in the effective delay penalty for the points $S > C/a$ (128K) in curves 1M and higher. In all these points there are R/S cache lines mapping to a single set and as S increases fewer lines are mapped to the set. A similar argument to that given above applies, except that now the exponent is $R/S - 1$. We can see that the second point from the right (identified by symbol §), which corresponds to stride $S = R/4$, should also have an effective delay penalty 7/8. Figure 3 shows that the miss penalty drops to the same value of curve 512K.

The previous discussion illustrates how effective the P^3 diagrams are in capturing the complex performance space exhibited by shared memory machines.

5.5. KSR1 Single Ring, Two Nodes Performance Results

It is fairly expensive to use data that resides in another node on the same ring in the KSR1. Figure 4 (*see also Figures C-3 and C-4 shown on the color plate page*) shows a set of curves for read-

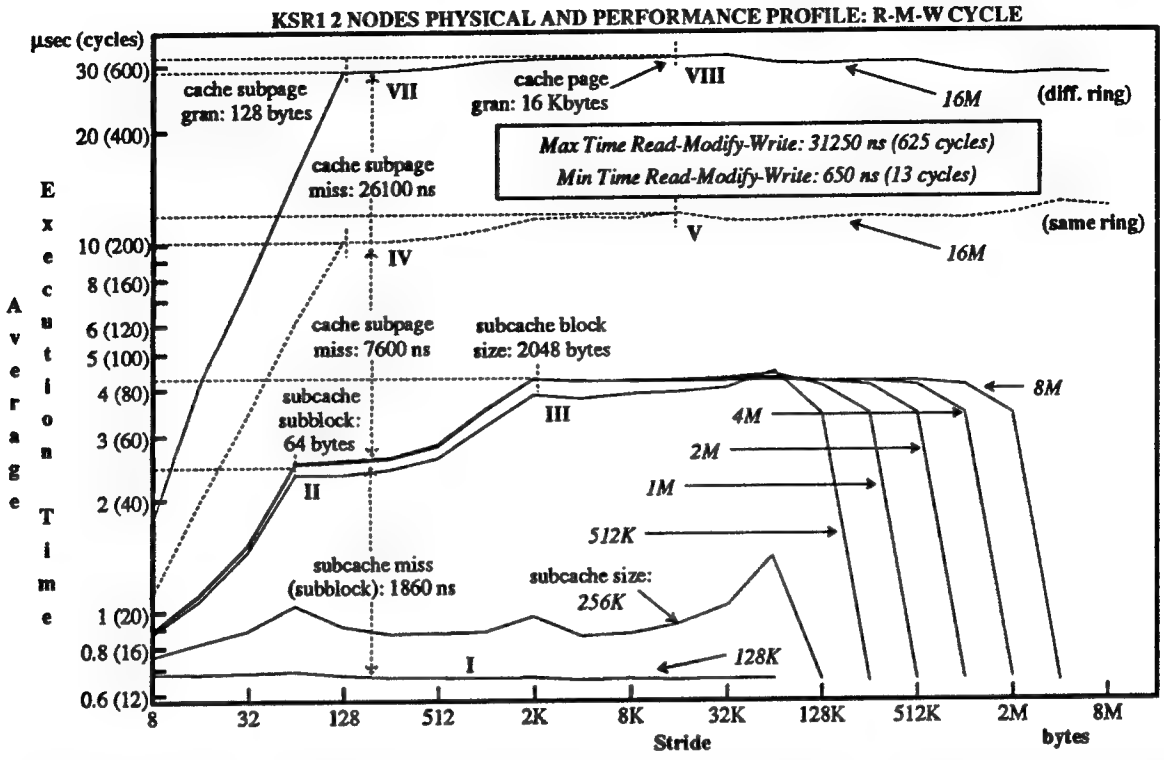

Figure 5: Physical and performance profiles of the KSR1 subcache and cache structures obtained using 2 competing nodes running R-M-W based experiments. Each regime identifies the mean execution time for a particular combination of subcache and cache miss penalties. For array sizes smaller then 16MB, there is no contention between the nodes.

modify-write (as was the case for Figure 2), when the data sets are as large or larger than the local cache. The 32 MByte line is particularly interesting. Not all the data can fit in the local cache at one time, since some space is needed for the program, and perhaps for the operating system or other pages for which the executing node is the home cache. But clearly there is enough reuse of blocks that even at large strides (8K to 2M), the average cost of the memory accesses is somewhat less than the cost of large strides for larger data sets.

As we mentioned in section 2, the KSR1 local cache is 32 MBytes and 16-way associative. However, some pages are "wired" by the operating system, so they cannot be selected as victims. In addition, the local cache acts as home of some fraction of the user's pages. Hence, the effective cache size that a program sees is significantly less than the 32 MBytes and the effective set associativity is less than 16.

Both of these characteristics are clearly present in Figure 4. For example, the 32*M* curve, which in principle should fit in the cache, clearly shows the presence of page misses with replacement. If the entire 32 MBytes were available, the curve's shape should be the same as the 16 MB*yte* curve. The reason why the curve reaches its highest point between strides 16K and 64K and then drops, is because the total number of pages touched by a particular experiment is constant for all strides less than 16K and then it decreases in proportion to the stride.

With respect to the set associativity, the three rightmost points of all curves having regions greater than 16 MBytes indicate that the cache associativity is 8-way. We can see this by noting that when the larger curves have a stride of 1/8 their size (e.g. 4MB stride in 32MB data set), their access time drops back to the subcache block miss level. This occurs because we are referencing only 8 different local cache pages and they map to a single set in the local cache. This value is less than the expected 16-way. Because our experiments change *R* and *S* only in powers of two, we detect 8-way associativity, while its real value can be any number between 8 and 16. We have performed more detailed experiments and have found that the effective associativity varies from set to set in the range from 3 to 12.

Finally, from Figure 4, we observe that the cost of removing and replacing a page in the cache is quite large (about 500 µsec). It should be borne in mind that one or more blocks in each page being replaced have been modified (are in the exclusive state), and must be evicted before a new block of the new page can be retrieved. This is a form of page swapping or thrashing between memories in the same ring, and does not require that the pages being evicted be written to disk.

5.6. KSR1 Two Rings, Two Nodes Performance Results

So far, we have discussed the performance of a single node in accessing data, though the data may reside on more than one node. We now turn our attention to the case in which multiple nodes are writing to the same set of data. Figure 5 (*see also Figures C-1 and C-2 shown on the color plate page*) shows an experiment in which two nodes in the same ring access data. This figure is like Figure 3, but with two additional curves labeled "16M", for a 16 MByte data set. Two processes on different nodes are simultaneously

Summary of the KSR1 Micro Benchmark Results Using One and Two Nodes

Regime	Misses				Rings?	Evict dirty page	R-M-W Cycle Iteration Time		
	Sub Cache		Local Cache				Total Time	Residual Time	
	subblock	block	subpage	page			time	time	Miss
I	no	no	no	no	n.a.	n.a.	0.65 µs	—	baseline
II	yes	no	no	no	n.a.	n.a.	2.50 µs	1.85 µs	subblock
III	yes	yes	no	no	n.a.	n.a.	4.20 µs	1.70 µs	block
IV	yes	no	yes	no	same	n.a.	10.10 µs	7.60 µs	subpage
V	yes	yes	yes	no	same	no	11.80 µs	1.70 µs	block
VI	yes	yes	yes	yes	same	yes	520.00 µs	508.00 µs	page
VII	yes	yes	yes	no	diff	n.a.	28.60 µs	26.10 µs	subpage
VIII	yes	yes	yes	yes	diff	no	31.10 µs	2.50 µs	block

Table 2: Mean execution time for a single read-modify-write iteration. Each regime represents a combination of *R* and *S* producing a particular pattern of misses to a subset of the memory hierarchy. Column "Evict dirty page" shows the delay involved in moving a dirty page out of a cache after a cache miss. The dirty page has to be sent back to the "home" node.

accessing the data set in read-modify-write mode, for different strides. In one case, the two nodes are on the same ring, and in the second case, they are on different rings. The figure shows that the cost of a miss in the local cache is about 7.6 µsec (at stride 128, the size of a subpage, every read misses), if the subpage is in another cache on the same ring. Note that the total cost of the access is the sum of all the misses, about 10.1 µsec.

The cost of accessing smaller data sets will be the same as shown in the 16M curve, since the two processors will be invalidating each others' cache subpages. The experiment was designed so that the starting point for each processor was separated by 1/2 the size of the data set (i.e., 8 megabytes apart). In this way, the processors are not competing for the same data at the same time except at large strides.

From the curve for the case of two processes located in nodes on different rings, the subpage miss penalty is 27.8 µsec. The machine used for the experiment had two ring:0 rings interconnected by a ring:1 ring. We assume that there were only two ring interfaces in ring:1. If traversal time in ring:0 is about 7.5 µsec for 33 nodes (including the ring interface), then about 13 µsec are consumed in the ring interfaces and in traversing ring:1. Since the per link data rate in ring:1 is the same as ring:0 for this machine, the cost of the directory operations and ring insertions would appear to be consuming most of this time.

The results displayed in Figures 4 and 5 show eight performance regimes (indicated by roman numerals in the figures). Each regime is determined by the number the misses it triggers in a number of enclosing levels of the hierarchy. Different regimes also identify the locations from which a miss can be satisfied.

Regime I (Figure 5) represents the baseline time; the case when no misses are triggered by the micro benchmark. Regime II adds to the baseline the delay due to subcache subblock misses, while regime III includes both subblock and block miss delays. Regimes IV and VII contain the effect of local cache subpage misses. The first captures the case when the miss is satisfied by a node in the same ring, while the second represents reading the data from a remote ring. In all regimes, except VI, the region of data covered by the micro benchmarks is less than the size of the local cache. Hence, all subpage misses occurring in these regimes are only the result of mutual invalidations between the nodes, because both nodes need exclusive rights over the subpage.

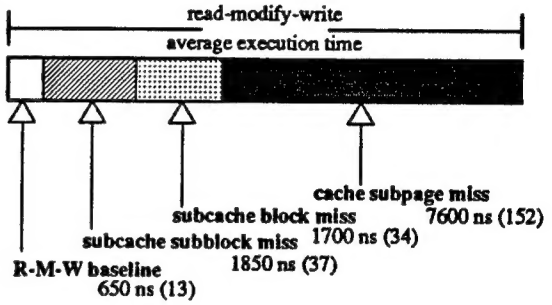

Figure 6: Components of regime V (ring:0 latency).

In regime VI, on the other hand, satisfying a miss requires either evicting a dirty page if the micro benchmark is based on the R-M-W cycle or detecting that the page is not dirty and just dropping it if it is based on the R-U cycle. In the former case the complete dirty page has to be sent to the "home" node. In both situations there is a significant extra penalty involved. The results just discussed are summarized in Table 2.

Figure 6 shows the component times of a memory access in regime V.

6. Communications Performance of the KSR1

The performance of the interconnection network has a significant effect on the overall performance of parallel computations and greatly affects the granularity achievable. The experiments reported in the previous section which measured the performance of the memory hierarchy were carefully designed to minimize the effects of loading of the communications network. For real applications, both memory performance and communications network performance will affect the overall rate of computation. In this section, we report on our experiments to investigate the performance of the KSR1 ring interconnection network.

The experiments are similar to our memory experiments. A single shared array is accessed by several nodes. The array is divided into equal portions, with each portion accessed in a read-modify-

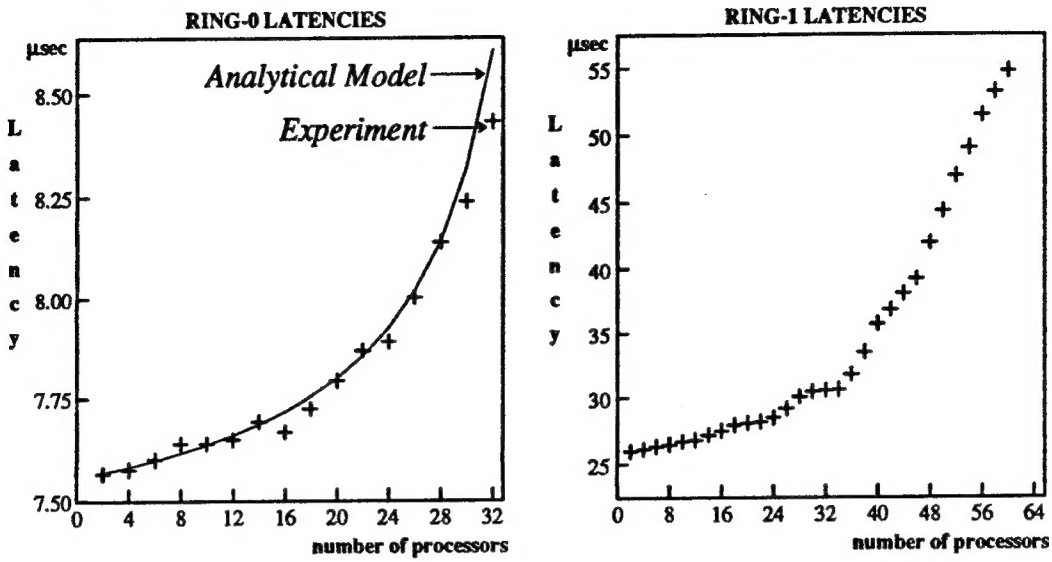

Figure 7: Communication latency as a function of the number of nodes communicating.

write cycle by only two nodes. The placement of the nodes and the assignment of portions of the array to nodes is carefully designed so that all nodes execute their portion of the experiment in in about the same amount of time. (This requires care; it is easy to construct experiments in which the performance of some of the nodes is much worse than other nodes, even though all are doing the same amount of work).

6.1. Contention on a Single Ring: Analytic Model

Here we present a simple model for the extra delay in latency due to contention in the ring in the case when all communication is local to ring:0. We then compare the model against experimental results. In our experiments, the time per iteration t_{iter} can be broken into two components: time of computation (t_{comp}) and time for communication (t_{comm}). Term t_{comm} has two additional components: the communication time without contention ($t_{no-cont}$) and the extra delay due to contention ($t_{penalty}(n_{nodes})$). The later term is a function of the number of communicating nodes (n_{nodes}). Let n_{slots} be the number of slots in ring:0. On the average, when a node wants to drop a packet into the ring, there are $(n_{nodes}-1)\cdot t_{no-cont}/t_{iter}$ other messages occupying slots. The probability that a random slot is empty is given by

$$p_{empty} = 1 - \frac{(n_{nodes}-1)t_{no-cont}}{n_{slots} \cdot t_{iter}}.$$

Now, the number of consecutive occupied slots passing through a node before an empty slot is found follows the geometric distribution with parameter p_{empty}. Hence, the expected number of consecutive occupied slots is $(1-p_{empty})/p_{empty}$[2]. Given that the time between successive slots is $t_{no-cont}/n_{slots}$, we can compute the expected extra delay in latency due to contention as

$$t_{penalty}(n_{nodes}) = \frac{t_{no-cont}}{n_{slots}} \left[\frac{1-p_{empty}}{p_{empty}} \right]. \quad (3)$$

Eq. (3) does not apply to the case when nodes in different rings communicate. Unfortunately, we do not have enough information about ring:1 and the interfaces between ring:0 and ring:1 to produce a realistic analytical model in this case.

6.2. Experimental Results

The graphs in Figure 7 shows the experimental values for $t_{comm} + t_{penalty}(n_{nodes})$ for various numbers of nodes. The figure labeled "RING-0 LATENCIES" also shows the latency predicted by eq. (3) in the single ring case.

The ring-0 results in Figure 7 clearly show that in the case of a single ring, even when the latency tends to increase with the number of nodes, this increase is relatively modest. In fact the total increase in latency going from 2 to 32 nodes is less than 15% of the original t_{comm}. There is a small error between the analytical and experimental results which increases with the number of nodes. The maximum error observed is less than 11%[3]. This is because eq. (3) overestimates $t_{penalty}$ by assuming that t_{iter} is independent of the number of nodes in the experiment. In actuality the time per iteration is given by

$$t_{iter}(n_{nodes}) = t_{comp} + t_{no-cont} + t_{penalty}(n_{nodes}).$$

Considering this new term in (1) reduces the error between the experimental and analytical results to approximately less than 5 percent.

The contention penalty when node communication requires sending messages through ring:1 shows a more interesting behavior. Figure 7.b distinctly shows two performance regimes, one for less than 32 nodes and another for more than 32 nodes. In the former case increasing the number of nodes by one, on the average

[2] The mean number of slots passing through a node, including the empty one, is given by

$$\frac{1}{p_{empty}} = \frac{1-p_{empty}}{p_{empty}} + 1.$$

[3] The 11% error in $t_{penalty}$ represents only a 2% error in terms of the total communication latency.

increases latency by 152 ns (~ 3 cycles), while in the later case, the increment is as large as 1000 ns (~ 20 cycles). A simple model based on linear fit of both regimes gives the following formulas:

$$L(n) = \begin{cases} 152\,\text{ns} \times n + 25200\,\text{ns} & \text{for } n \le 32 \\ 1004\text{ns} \times (n-32) + 27500\,\text{ns} & \text{for } n > 32. \end{cases} \quad (4)$$

with respective correlation coefficients of 0.9713 and 0.9116. When the number of active nodes increases from two to 32 and 60, then eq. (4) gives a relative increase of 21% and 110% respectively in the total latency.

From the data presented in Figure 7, we can estimate the data rate per node under various communications loads. The conflict-free rate for a node is about 17 megabytes/second (a 128 byte subpage every 7.6 μsec). As the load gets heavy within a single ring, with the ratio between requests for data (cache misses) and computation of our experiments, the rate declines to about 15 megabytes/second when all 32 nodes are generating requests. When data moves between rings, rates are much lower. The range is 5 megabytes for one node without contention to about 4 megabytes when 32 nodes are active, and declining to less than 2.5 megabytes per second per node with a load of 60 nodes.

7. Related Work

Recently several other researchers have been investigating the performance of the KSR1. Boyd et. al [1] show a method of measuring communications performance on multiprocessors using a synthetic workload based on matrix multiplication of generated matrices. Other researchers [10, 9] have also reported on experiments to measure the performance effects of specific features of the KSR1. The communication and synchronization performance of the KSR1 has been analyzed by Dunigan [4]. Singh et. al [15] present performance results of several kernel codes and some of the SPLASH benchmark suite on the KSR1 and DASH machines. Finally, analytic model comparing the potential benchmark performance of NUMA and COMA machines has been developed by Hagersten [5].

8. Conclusions

Based on our measurements, it appears that the KSR1 ALLCACHE memory architecture should gracefully extend to a large number of processors. It indeed fulfills its promise of a scalable shared memory architecture. As with any parallel machine, the performance of parallel applications will depend on the degree and form of interactions between computations running on separate nodes of the machine. From our results, it is clear that there are types of shared access that are expensive, and that the programmer should be aware of the costs of accessing data, especially at larger strides, that reside on a different node from the accessing node.

The overall performance of any machine is a combination of its many features. The ALLCACHE memory is only one component of the KSR1. In addition to the processor, the ring of rings interconnect is a major element. The architecture is interesting and effective. Our results do not permit us to give a relative evaluation of the machine in comparison with other architectures, but our data will, we believe, help potential users in deciding whether to use the machine, and how to use it effectively.

9. Acknowledgements

We want to thank Tom Dunigan at Oak Ridge for giving us access to the KSR1 and to Eric Boyd who made useful suggestions.

10. References

[1] Boyd, E., Wellman, J.D., Abraham, S., and Davidson, E., "Evaluating the Communication Performance of MPPs Using Synthetic Sparse Matrix Multiplication Workloads", *Proc. of the 7th ACM Int. Conf. on Supercomputers*, Tokio Japan, July 1993.

[2] Bryant, C., *Personal communication*, April 1993.

[3] Cybenko, G., Kipp, L., Pointer, L., and Kuck, D., *Supercomputer Performance Evaluation and the Perfect Benchmarks*, University of Illinois Center for Supercomputing R&D Tech. Rept. 965, March 1990.

[5] Dunigan, T.H, "Kendall Square Multiprocessor: Early Experience and Performance", Oak Ridge National Laboratory Tech. Rept. No. ORNL/TM-12065, April 1992.

[6] Hagersten, E., Landin, A., and Haridi, S., "DDM -- A Cache-Only Memory Architecture", *Computer*, September 1992, pp. 44-54.

[7] Kendall Square Research, *KSR Parallel Programming*, KSR1 Documentation, February 1992.

[8] Lamport, L. "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", *IEEE Transactions on Computers*, Vol.C-28, No.9, September 1979, pp. 690-691.

[9] Lenoski, D., Laudon, J., Gharachorloo, Gupta, A., and Hennessy, J., "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor", *Proc. of the 17th Int. Symp. on Comp. Arch.*, May 28-31 1992, Seattle, Washington, pp. 148-159.

[10] Ramachandran, U., Shah, G., Ravikumar, S., and Muthukumarasamy, J., "Scalability Study of the KSR-1", 22nd Int. Conf. on Parallel Processing, St. Charles, August 1993.

[11] Rosti, E., Smirni, E., Wagner, T., Apon, A., and Dowdy, L., "The KSR1: Experimentation and Modeling of Poststore", *Proc. of the 1993 ACM Sigmetrics Conf. on Meas. & Modeling of Comp. Sys.*, Santa Clara, California, May 1993, pp. 74-85.

[12] Saavedra-Barrera, R.H., Smith, A.J., and Miya, E. "Machine Characterization Based on an Abstract High-Level Language Machine", *IEEE Trans. on Comp.* Vol.38, No.12, December 1989, pp. 1659-1679.

[13] Saavedra-Barrera, R.H., *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking*, Ph.D. Thesis, U.C. Berkeley, Tech. Rept. No. UCB/CSD 92/684, February 1992.

[14] Saavedra, R.H., Gaines, R.S., and Carlton, M.J., "Characterizing the Performance Space of Shared Memory Computers Using Micro-Benchmarks", USC Tech. Rept. No. USC-CS-93-547, July 1993.

[15] Singh, J.P., Truman, J., Hennessy, J., and Gupta, A., "An Empirical Comparison of the Kendall Square Research KSR-1 and Stanford DASH Multiprocessors", *Supercomputing'93*, November 1993.

[16] SPEC, "*SPEC Newsletter: Benchmark Results*", Vol.2, Issue 1, Winter 1990.

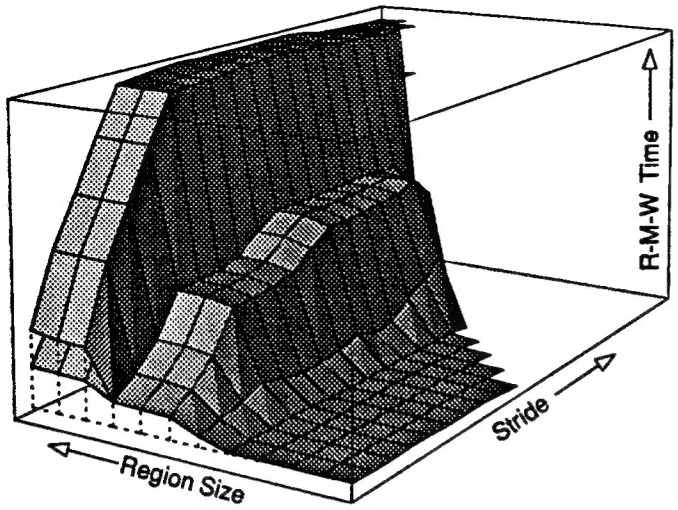

Fig. C-1: KSR1 2-Node Physical and Performance Profile (Fig. 5). The projection is taken from point {-2.5, -1.7, +0.5}.

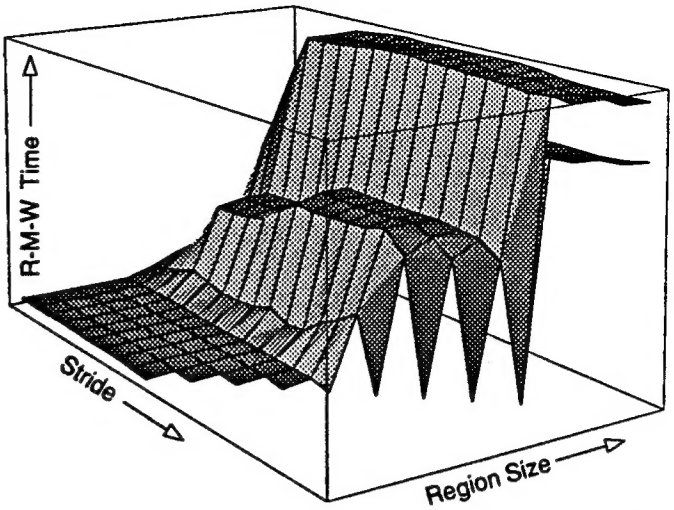

Fig. C-2: KSR1 2-Node Physical and Performance Profile (Fig. 5). The projection is taken from point {-2.5, +1.7, +0.5}.

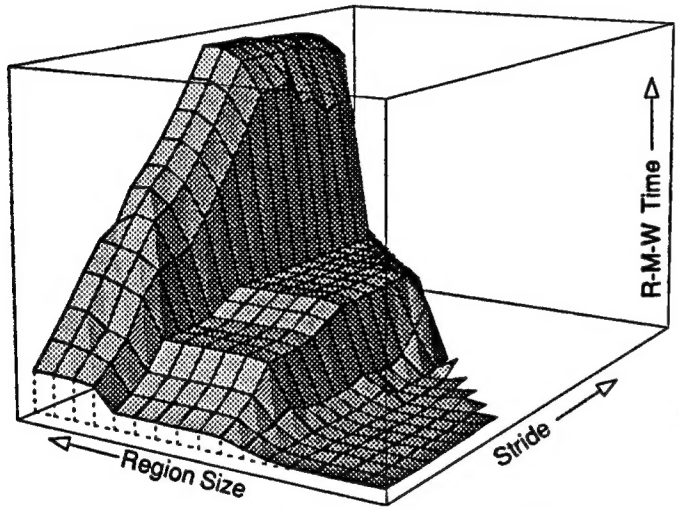

Fig. C-3: KSR1 1-Node Physical and Performance Profile (Fig. 4). The projection is taken from point {-2.5, -1.7, +0.5}.

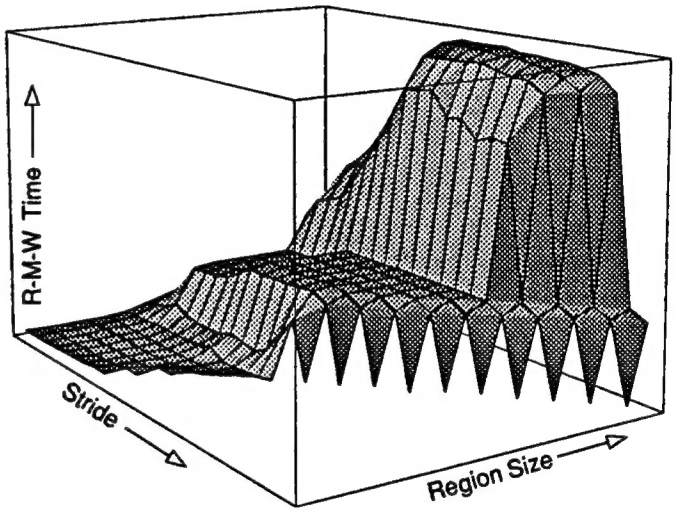

Fig. C-4: KSR1 1-Node Physical and Performance Profile (Fig. 4). The projection is taken from point {-2.5, +1.7, +0.5}.

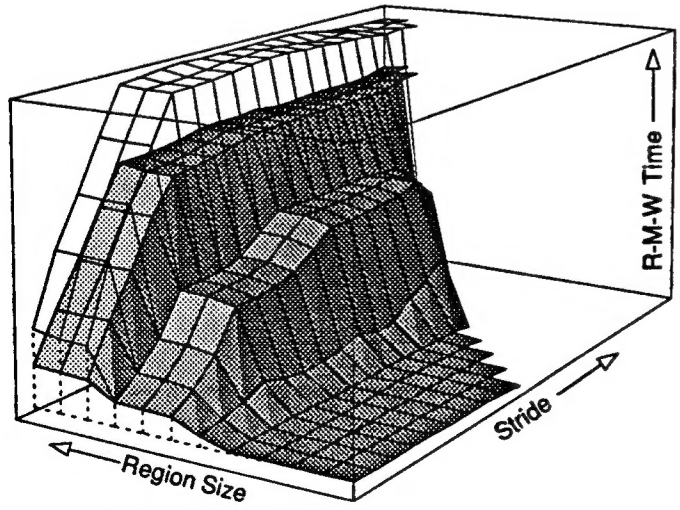

Fig. C-5: KSR1 2-Node Physical and Performance Profile (Fig. 5). Cache misses satisfied in local ring:0 are shown.

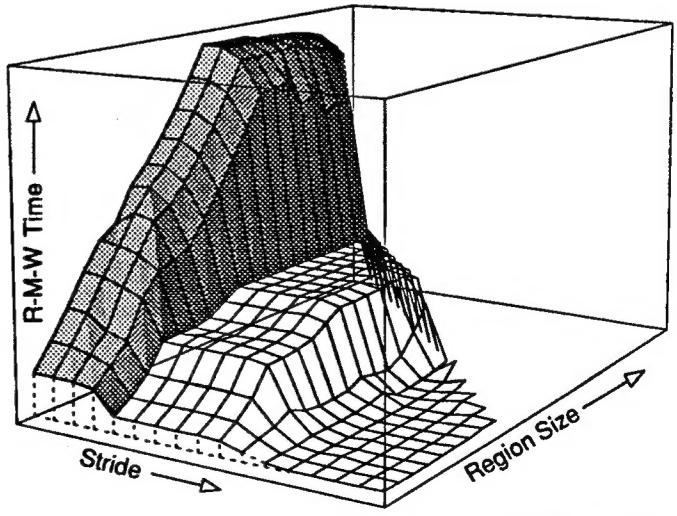

Fig. C-6: KSR1 1-Node Physical and Performance Profile (Fig. 4). The effect of misses with page replacement is shown.